CSE 344: Section 8 Parallel Planning MapReduce and Spark

November 16th, 2017

Administrivia

- HW6 (AWS) due next Tuesday
  - Queries take a while
  - Avoid doing everything at the last minute!

Distributed Query Processing

In this class, only **shared-nothing architecture** and **intra-operator parallelism**

Horizontal Data Partitioning:

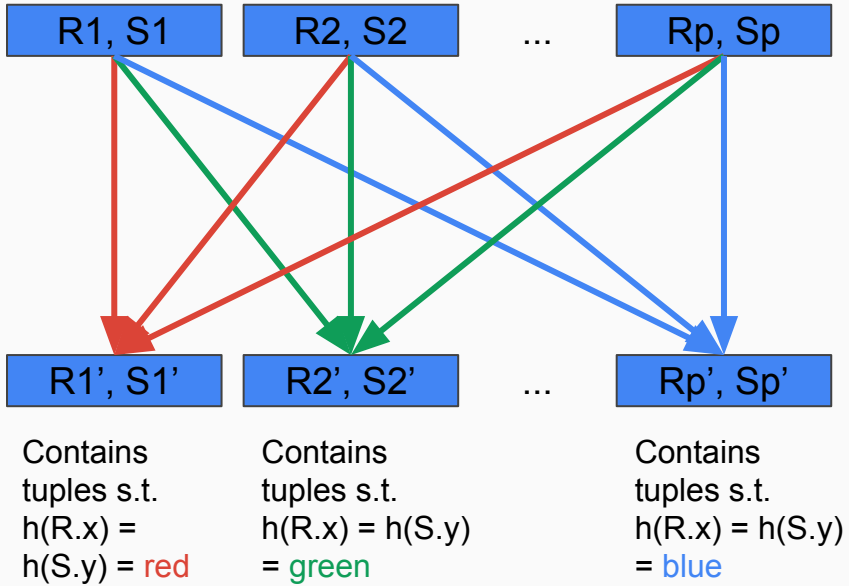
- Block Partition
- Hash partitioned on attribute A
- Range partitioned on attribute A

Partitioned Hash-Join Mechanism

We have p machines

We wish to join on some attribute (say R.x and S.y)

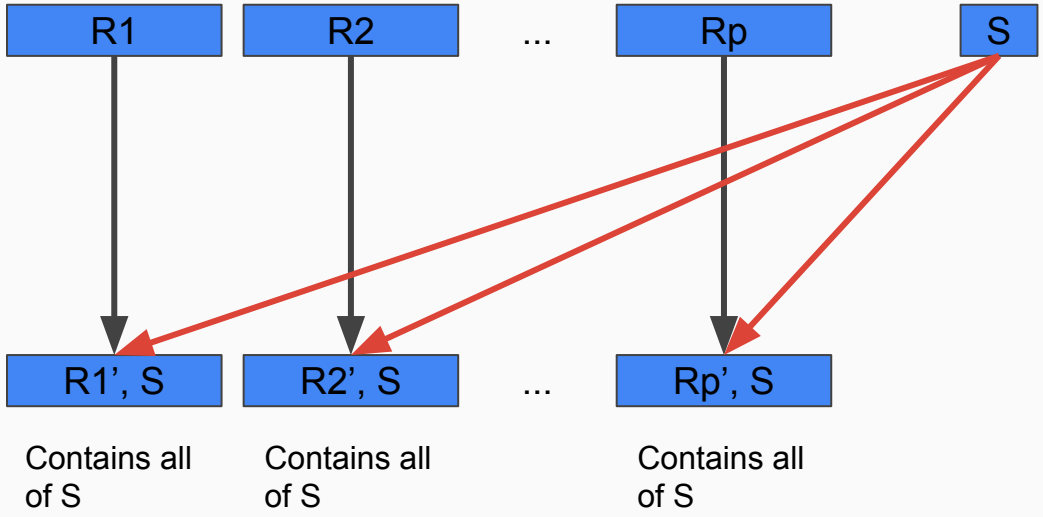
Call our hash function h(z)



Broadcast Join (Map-Side Join) Mechanism

We want to think about how to prevent sending all data through the network.

Take advantage of small datasets (meaning the whole dataset can fit into main memory)



Parallel Query Planning

Now consider network I/Os

- Only send tuples to other nodes when we need to
- Size of data for a particular relation is a good heuristic to determine if you want to pass it around

MapReduce

Distributed File System (DFS)

MapReduce Job:

- Map Task (EmitIntermediate)
- Reduce Task (Emit)

Fault Tolerance (replicated chunks, write intermediate files to disk)

Spark

Resilient Distributed Datasets (RDD)

High level commands:

- Transformations (map, reduce, join…) -> **Lazy**
- Actions (count, reduce, save...) -> **Eager**

Fault Tolerance (main memory and lineage)

Spark Objects for HW6

```
Row

RowFactory.create(Objects...)

Dataset<Row>

JavaRDD<Row>

JavaPairRDD<K, V>

Tuple2<>
```

you can leave the generics empty

Spark Methods for HW6

```
d.sql("SELECT ... FROM ...")
```
d must be a Dataset (HW6 Q1)

```
d.filter(t -> f(t) == true/false)
```

```
d.distinct()
```

```
d.map()
```
d must be a JavaRDD

```
d.mapToPair(t -> new Tuple2<>(K, V))
```

```
d.reduceByKey((v1, v2) -> f(v1, v2))
```
d must be a JavaPairRDD

Spark Demo